

TRIE

Handwritten Notes of Striver(TUF) Playlist

by: Aashish Kumar Nayak

NIT Srinagar



AASHISH KUMAR NAYAK



aashishkumar.nayak

TRIE SERIES

- L1. Implement TRIE | INSERT | SEARCH | STARTSWITH
- L2. Implement TRIE-2 | INSERT | CountWordSequalTo() | Countwords Starting with()
- L3. Longest word with All prefixed | complete string | Trie
- L4. Number of Distinct sub strings in a string | Trie
- L5. Bit prerequisites for TREE problems
- L6. Maximum XOR of Two Numbers in An Array
- L7. Maximum XOR with an Element from Array | Queries

TRIE

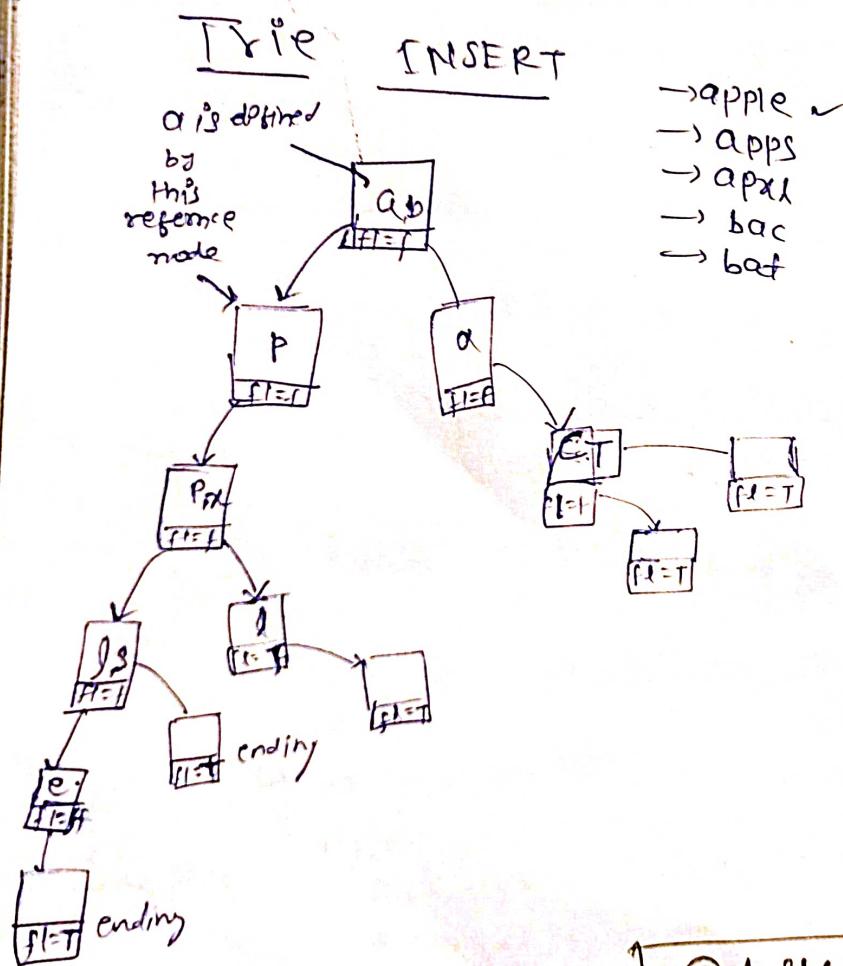
L-1 Implement TRIE | INSERT | SEARCH | STARTWITH

Trie data structure is used whenever we have to insert words then they will ask you if this particular word exists.

Start and is there any word whose prefix is this word.

5

- 1 hello // means insert hello
- 1 help // means insert help
- 2 help // means search help // output true
- 3 hel // means is there any word starts with hel // output true
- 2 hel // means search hel // output false.



Search functionality.

lets say some search of apps

we will traverse in trie, and if our last reference's flag is True then means we got that word, else return false.

Starts with functionality

let say \rightarrow ba
 \rightarrow apd

we will traverse in trie and if we are standing at a not null position means there is word with given prefix.

If we encountering a null during traversal of trie

means we didn't get any word with that prefix.

Code :-

Struct Node {

Node *links[26];

bool flag = false;

bool containkey (char ch)

{ return (links[ch - 'a'] != NULL); }

void put (char ch, Node *node)

{ links[ch - 'a'] = node; }

Node *get (char ch)

{ return links[ch - 'a']; }

void setEnd()

{ flag = true; }

bool isEnd()

{ return flag; }

} }

Class Trie {

private:

Node* root;

public:

Trie();

root = new Node();

void Insert(string word)

Node *node = root;

for (int i=0; i<word.size(); i++)

{ if (!node->ContainKey(word[i]))

{ node->put (word[i], new Node());

}

node = node->get (word[i]);

node->setEnd();

void bool search (string word)

Node *node = root;

for (int i=0; i<word.size(); i++)

{ if (!node->ContainKey (word[i]))

{ return false; }

node = node->get (word[i]);

return node->isEnd();

```
bool startsWith(string prefix)
```

```
{ Node* node = root;
for (int i = 0; i < prefix.size(); i++)
{
    if (!node->containsKey(prefix[i]))
        return false;
    node = node->get(prefix[i]);
}
return true;
}; }
```

```
int main()
```

```
{ int n = 5;
vector<int> type = {1, 1, 2, 3, 2};
vector<string> value = {"hello", "help", "help", "hel", "hel"};
Trie trie;
for (int i = 0; i < n; i++)
{
    if (type[i] == 1)
        trie.insert(value[i]);
    else if (type[i] == 2)
        if (trie.search(value[i]))
            cout << "true" << "\n";
        else
            cout << "false" << "\n";
    else
        if (trie.startsWith(value[i]))
            cout << "true" << "\n";
        else
            cout << "false" << "\n";
}
```

Implement Trie - II

implement functionalities

✓ countWordsEqualto "word"

✓ countWordsStartingWith "word"

✓ Erase ("word")

1

6

insert Sumsung

insert sum sung

insert vivo

erase vivo

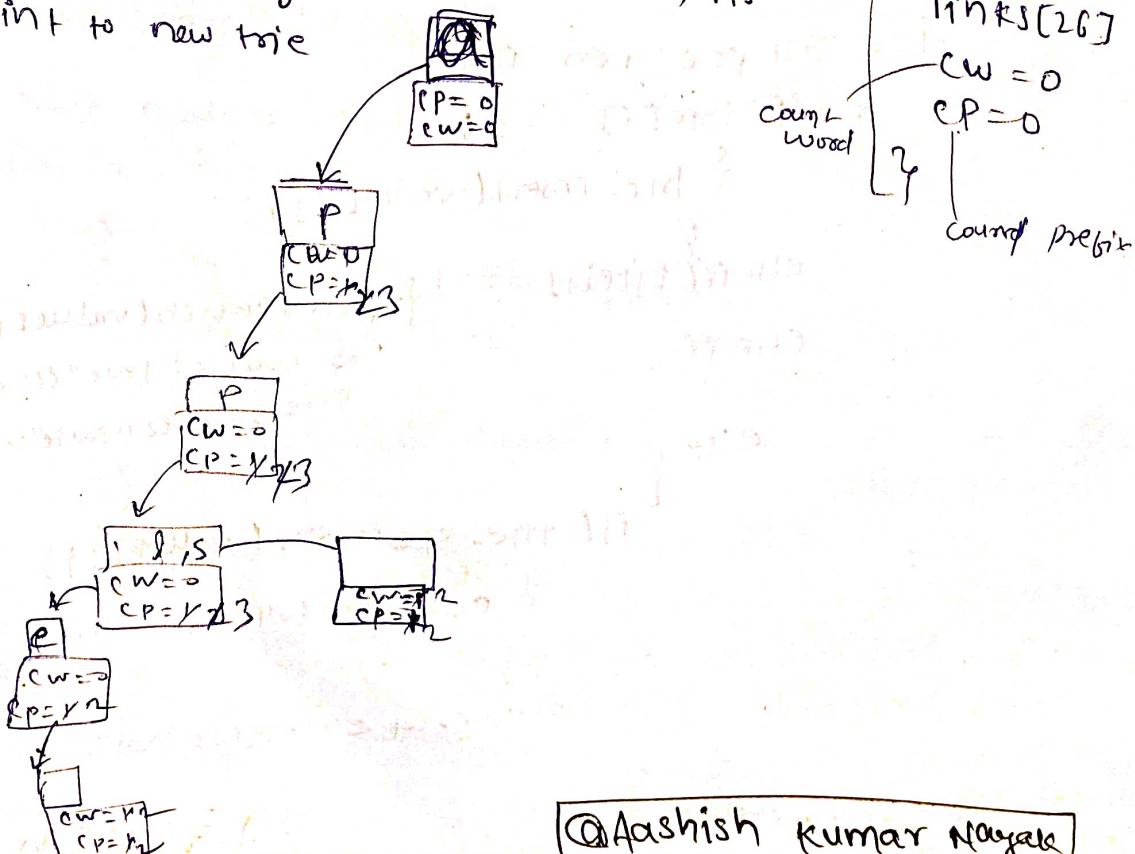
countWordsEqualto sumsung

countWordsStartingWith vi

// output - 2

// output - 0

insert @ and having
point to new trie



ex-1

Reduce the prefix and also at the end
also reduce count word.

Code :-

Struct Node {

Node *links[26]; int cntEndwidth; freq;

int cntprefix = 0;

bool containsKey(char ch);

{ return links[ch - 'a'] != NULL; }

Node *get(char ch);

{ return links[ch - 'a']; }

void put(char ch, Node *node);

{ links[ch - 'a'] = node; }

void increaseEnd();

{ cntEndwidth++; }

void increasePrefix();

{ cntprefix++; }

void deleteEnd();

{ cntEndwidth--; }

int getEnd();

{ return cntEndwidth; }

int getprefix();

{ return cntprefix; }

};

```

class Trie {
    private:
        Node *root;
    public:
        Trie() {
            root = new Node();
        }
        void insert(string word) {
            Node *node = root;
            for(int i=0; i<word.length(); i++) {
                if(!node->containsKey(word[i])) {
                    node->put(word[i], new Node());
                }
                node = node->get(word[i]);
                node->increasePrefix();
            }
            node->increaseEnd();
        }
        int countWordsEqualTo(string word) {
            Node *node = root;
            for(int i=0; i<word.length(); i++) {
                if(node->containsKey(word[i])) {
                    node = node->get(word[i]);
                } else {
                    return 0;
                }
            }
            return node->getEnd();
        }
}

```

```

int countwordsStartingWith(string &word)
{
    Node *node = root;
    for(int i=0; i<word.length(); i++)
    {
        if(node->containsKey(word[i]))
        {
            node = node->get(word[i]);
        }
        else
        {
            return 0;
        }
    }
    return node->getPrefix();
}

void erase(string &word)
{
    Node *node = root;
    for(int i=0; i<word.length(); i++)
    {
        if(node->containsKey(word[i]))
        {
            node = node->get(word[i]);
            node->reducePrefix();
        }
        else
        {
            return;
        }
    }
    node->deleteEnd();
}

int main()
{
    Trie T;
    T.insert("apple");
    T.insert("apple");
    T.insert("apps");
    T.insert("apps");
}

```

String word1 = "apps";

cout << "Count words equal to " << word1 << endl; T.countWordsEqualto(word1) << endl;

String word2 = "abc";

cout << "Count words Equal to " << word2 << endl; T.countWordsEqualto(word2) << endl;

String word3 = "ap";

cout << "Count words starting with " << word3 << endl; T.countWordsStartingWith(word3) << endl;

String word4 = "app";

cout << "Count words starting with " << word4 << endl; T.countWordsStartingWith(word4) << endl;

To erase (word1);

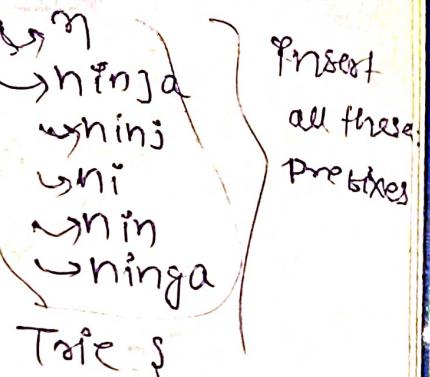
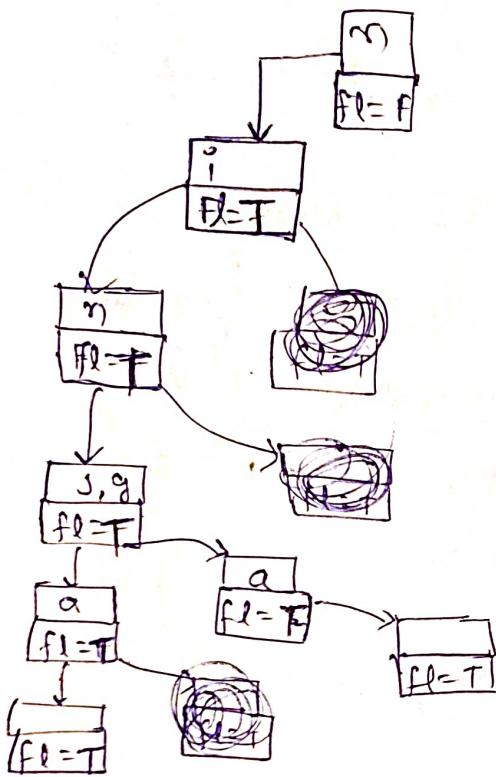
cout << "Count words equal to " << word1 << endl; T.countWordsEqualto(word1) << endl;

return 0;

Longest word with All Prefixes

complete strings
These should be C.S.

Complete string



arr[26]
bool flag

complete string

n

T.C.

$O(N) \times O(\text{len})$

S.C.

$O(N \times \text{len})$

string length

Code :-

struct Trie {

Node* links[26];

bool flag = false;

bool containsKey(char ch)

{ return links[ch - 'a'] != NULL; }

Node* get(char ch) { return links[ch - 'a']; }

void put(char ch, Node* node)

{ links[ch - 'a'] = node; }

```

Void setEnd()
{
    return flag = true;
}

bool isEnd()
{
    return flag;
}

```

Class Trie

```

{
    private : Trie root;
    public :
        Trie()
        {
            root = new Node();
        }

    public : void insert(string word)
    {
        Node * node = root;
        for(int i=0; i<word.length(); i++)
        {
            if(!node->containsKey(word[i]))
            {
                node->put(word[i], new Node());
            }
            node = node->get(word[i]);
        }
        mode->setEnd();
    }
}

```

Public : bool checkPrefixExists(string word)

}

→ Caashish Kumar Nayak

```
String completeString (int n, vector<string> &a)
```

```
{ Trie trie;
```

```
for (auto &it : a)
```

```
{ trie.insert(it);
```

```
String longest = " ";
```

~~for (auto &it : a)~~~~{ IF (checkIfPrefixExists(it))~~~~{ if (it.length() > longest.length())~~~~{ longest = it;~~~~}~~~~else if (it.length() == longest.length() &~~~~{ it < longest)~~~~{ longest = it;~~~~}~~~~}~~~~IF (longest == " ") return "None";~~

```
} return longest;
```

```
public : bool checkIfPrefixExists(string word)
```

```
{ bool f1 = true;
```

```
Node * node = root;
```

```
for (int i = 0; i < word.length(); i++)
```

```
{ if (node->containsKey(word[i]))
```

```
{ node = node->get(word[i]);
```

```
if (node->isEnd) == false) return false;
```

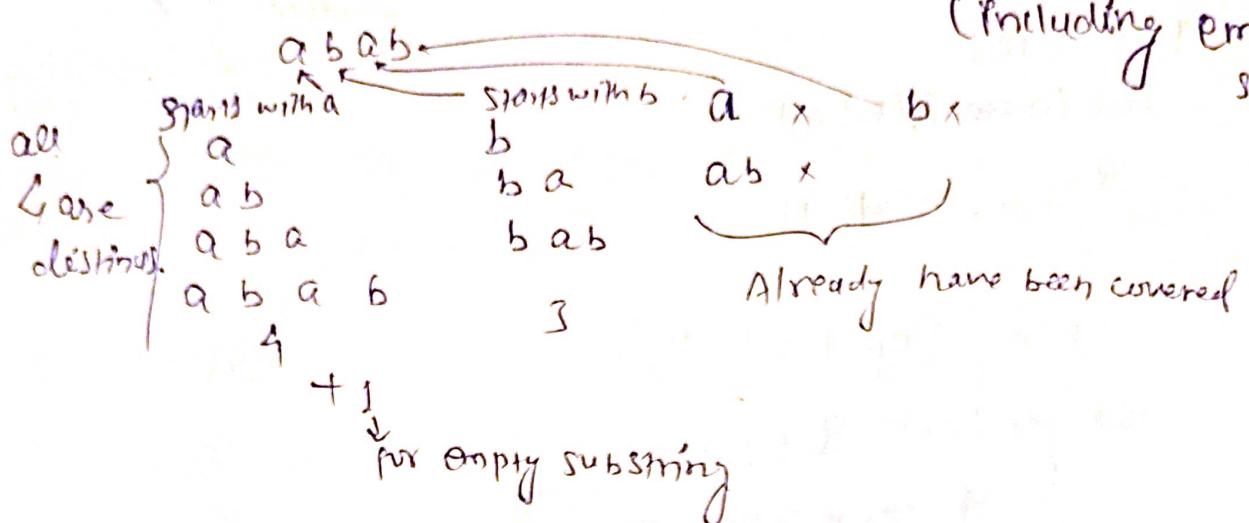
```
{ return false;
```

```
}
```

```
return true;
```

Number of different substrings in a string

(including empty string)



$$4+3+1 = 7+1=8$$

Set <string> st;

for (i=0 : $\rightarrow n-1$)

T.C.

$O(N^2)$

for (j=i $\rightarrow n-1$)

st = str + st[j]

st.add(st); — $\log N$

Print (st.size()) \rightarrow distinct substring

all of the substring generated
are distinct.

T.C. $O(N^2 \times \log M)$

length of
substring

aba
abab

S.C.

$O(4)$ in set

min $\rightarrow 1$

max $\rightarrow n$

but in set

S.C.

$O(3) + O(4)$

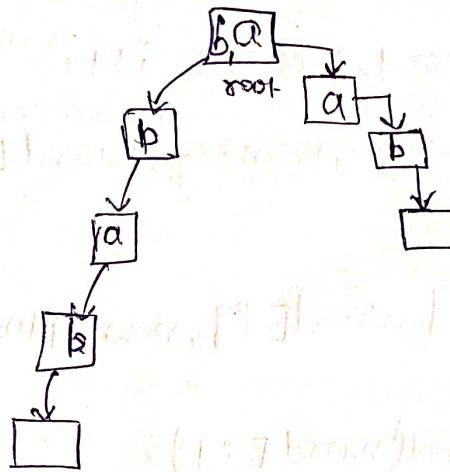
$\approx (n^2 / 2) \times n^2$

$n^2 \times n^2$

$9 \rightarrow O(1)$
 $ab \rightarrow O(2)$
 $aba \rightarrow O(3)$

Set

no. of
elements in
set.



abab

tries

arr[26];

}

++H +

++H + $\frac{3}{7}$ + empty

= 8
distinct
substring

T.C. $\rightarrow O(N^2)$

S.C. \rightarrow

Code:

```
class Node {
    Node *links[26];
    bool containsKey(char ch) {
        return links[ch - 'a'];
    }
    void put(char ch, Node* node) {
        links[ch - 'a'] = node;
    }
    Node* get(char ch) {
        return links[ch - 'a'];
    }
};
```

```
int countDistinctSubstrings(string &s)
```

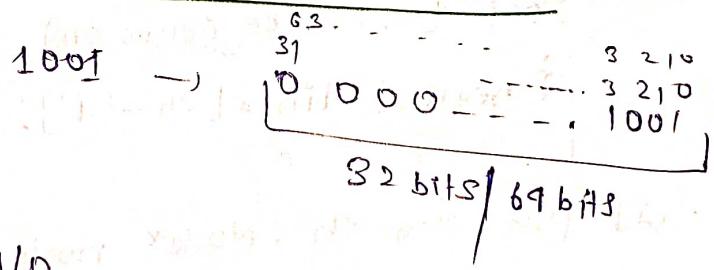
```
{ int cont = 0;
    Node* root = new Node();
    for (int i = 0; i < word.size(); i++)
        Node* node = root;
```

```

for (int j = i; j < word.size(); j++) {
    if (!node->containsKey(word[i])) {
        cout << " ";
        node->put(word[i], new Node());
    }
    node = node->get(word[i]);
}
return count;
}

```

Bit Prerequisites for Tries

9 → 1001 →  32 bits / 69 bits

XOR

$1 \wedge 0 = 1$ } XOR of different bits gives you 1
 $0 \wedge 1 = 0$ }
 $0 \wedge 0 = 0$ } XOR of same bits gives you 0.
 $1 \wedge 1 = 0$ }

If there are even no. of 1's → 0

If there are odd no. of 1's → 1

How to check if a bit is set or not

9 → 0000 - - - 01001 3rd bit → 1

0 00 - - - 0001 0 001 $\gg 3$ ($num \gg 3$) & 1

If (num >> i) & 1 b = 0) ✓

else ($\mu\mu\eta\eta$)_i & $\eta_1 = =) \chi$

How do we turn on a bit?

$$\begin{array}{r}
 9 \rightarrow 0\ 0\ 0 \ldots 1 \boxed{0} \ 0\ 1 \\
 0\ 0\ 0 \ldots 0\ 0\ 0\ 1 \\
 \hline
 0\ 0\ 0 \ldots 0\ 1\ 0\ 0 \quad (1 < 2) \\
 \hline
 1\ 1\ 0\ 1
 \end{array}$$

num1(1<*i*)

MAXIMUM XOR OF TWO NUMBERS in an Array

Given an array of numbers, & a number x ,
find the max value of $(arr[i] \wedge n)$.

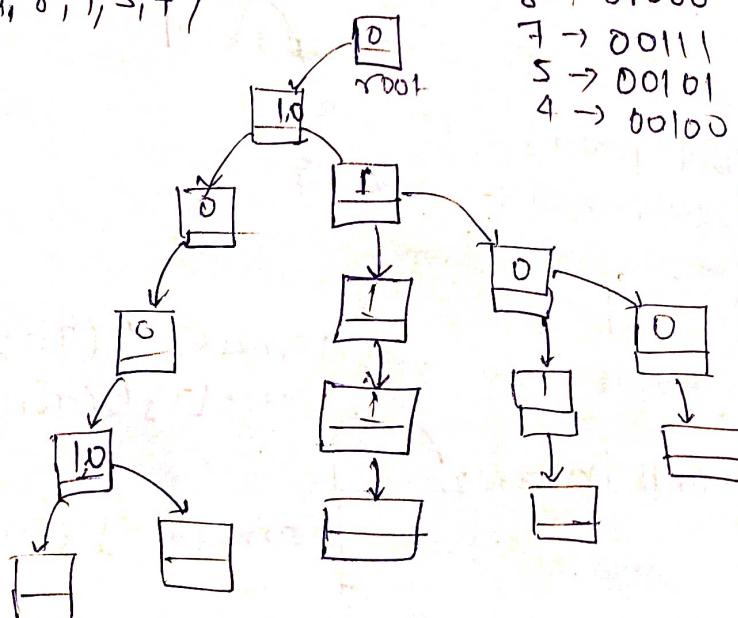
→ Insert all the numbers into the trie.

→ Take \star & find the binary bits)

max no. from array where (no n₁) ↑↑

$9 \rightarrow (0\ 000\ \dots\ 01001) \rightarrow$ we will store it as a string
 let's understand in terms of 5 bits

9, 8, 7, 5, 4 }



$9 \rightarrow 01001$
 $8 \rightarrow 01000$
 $7 \rightarrow 00111$
 $5 \rightarrow 00101$
 $4 \rightarrow 00100$

Nodes

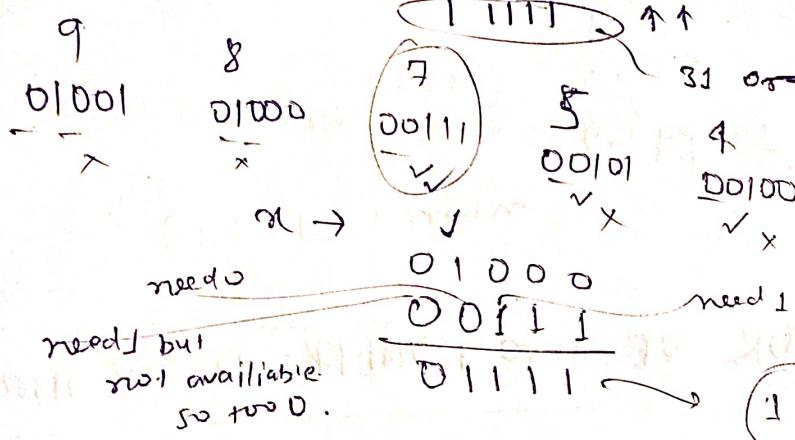
Now figure out which is
the largest

Intuition of Maximising XOR

$\lambda = 8^{\text{th}}$ arm — sit as high as possible

$$\begin{array}{r} 2^4 \\ \sqrt[2]{2212} \\ \hline 0 \end{array}$$

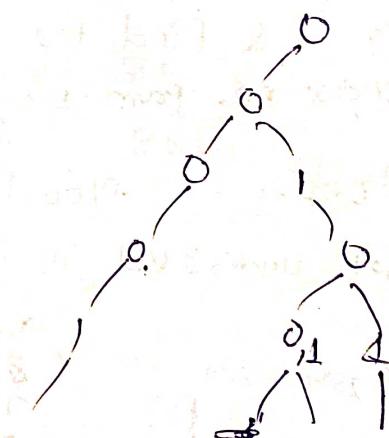
$$a[i] \rightarrow \underline{1} 0111$$



$$x = 8$$

9th 3rd, nd
0 1000

$$\begin{array}{r} 100111 \\ \hline 010011 \end{array}$$



ang[]

↓
χ

$$(x^y)^z = ? \uparrow \uparrow$$

arr₁ $\xrightarrow{\text{insert in}} T[0]$

$x \rightarrow$ all the elements in ans_2'

ans[0] ^ (T₀)
ans[1] ^ (T₁)

$$a \approx 2[m-1]^n(T_{\text{rise}})$$

maximum
of these.

```
struct Node {
```

```
    Node * links[2];
```

```
    bool containsKey(int ind)
```

```
    { return (links[ind] != NULL); }
```

```
}
```

```
    Node * getKey(int ind)
```

```
    { return links[ind]; }
```

```
    void put(int ind, Node * node)
```

```
    { links[ind] = node; }
```

```
}
```

```
class Trie {
```

```
private: Node * root;
```

```
public:
```

```
    Trie()
```

```
    { root = new Node(); }
```

```
}
```

```
public:
```

```
    void present(int num)
```

```
    { Node * node = root;
```

```
        for (int i = 31; i >= 0; i--)
```

```
            { int bit = (num >> i) & 1;
```

```
                if (!node->containsKey(bit))
```

```
                    { node->put(bit, new Node()); }
```

```
                node = node->getKey(bit);
```

```
}
```

```
public:
```

```
    ~Trie() { present(0); }
```

```
    Node * root;
```

(class Trie)

root

insert()

getMax(x)

}

T.C. $\rightarrow O(N \times 3^2)$

+

$O(M \times 3^2)$

go and arr

```

int maxNum = 0;
for (int i = 31; i >= 0; i--) {
    int bit = (num >> i) & 1;
    if (node->containsKey(!bit)) {
        maxNum = maxNum | ((1 << i));
        node = node->get(!bit);
    } else {
        node = node->get(bit);
    }
    return maxNum;
}

```

```

int maxXOR(int n, int m, vector<int>& arr1, vector<int>& arr2) {
    Trie trie;
    for (int i = 0; i < n; i++) {
        trie.insert(arr1[i]);
    }
    int maxi = 0;
    for (int i = 0; i < m; i++) {
        maxi = max(maxi, trie.findMax(arr2[i]));
    }
    return maxi;
}

```

```

int main() {
    vector<int> arr1 = {6, 8};
    vector<int> arr2 = {7, 8, 2};
    int n = 2, m = 3;
    cout << maxXOR(n, m, arr1, arr2) << endl;
    return 0;
}

```

Maximum XOR with an Element From Array

2
5 2
0 1 2 3, 4
Queries
 { 1 3
 5 6
 1 1
 1
 1 0 }

XOR $\downarrow \wedge$ (no. in array which is lesser than 3)
and return max.

Brute force

for ($i=0 \rightarrow q-1$)

```

  {
    xi = queries[i][0];
    ai = queries[i][1];
    Max_XOR = -1;
  
```

for ($j=0; j < n; j+1$)

```

  {
    if (arr[j] <= ai)
      Max_XOR = max (Max_XOR, arr[j] ^ xi);
  }

```

ans = .
Max_XOR.

Trie Method

Initially sort the array.

arr[] \rightarrow {1, 3, 2, 5, 4}

less than 1 put in trie

{3, 4}, 0

trie

{5, 2}, 1

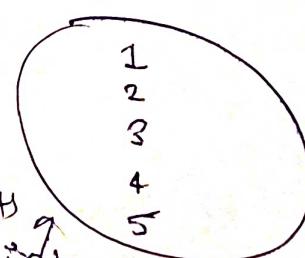
→

{2, 5}, 2

{3, 1}, 3

Now it gets converted into

[0 0 0 0] \rightarrow



1
2
3
4
5

previous problem.

get max(2)

Code

```

struct Node {
    Node* links[2];
    bool containsKey(int find) {
        return links[find] != NULL;
    }
    Node* get(int find) {
        return links[find];
    }
    void put(int ind, Node* node) {
        links[ind] = node;
    }
};

class Trie {
private:
    Node* root;
public:
    Trie() {
        root = new Node();
    }
public:
    void insert(int num) {
        Node* node = root;
        for(int i=31; i>=0; i--) {
            int bit = (num >> i) & 1;
            if(!node->containsKey(bit))
                node->put(bit, new Node());
            node = node->get(bit);
        }
    }
};

```

```

public:
    int findmax(int num)
    {
        Node* node = root;
        int maxnum = 0;
        for (int i = 31; i >= 0; i--)
        {
            int bit = (num >> i) & 1;
            if (node->containskey(!bit))
            {
                maxnum = maxnum | (1 << i);
                node = node->get(!bit);
            }
            else
            {
                node = node->get(bit);
            }
        }
        return maxnum;
    }

    vector<int> maxxorqueries(vector<int> &arr, vector<vector<int>> &queries)
    {
        vector<int> ans(queries.size(), 0);
        vector<pair<int, pair<int, int>>
        vector<pair<int, pair<int, int>> offlinequeries;
        sort(arr.begin(), arr.end());
        int i = 0;
        int n = arr.size();
        Trie trie;
        for (auto &it : offlinequeries)
        {
            while (i < n && arr[i] <= it.first)
            {
                trie.insert(arr[i]);
                i++;
            }
            if (i == 0) ans[it.second.second] = trie.findmax(it.second.first);
            else ans[it.second.second] = -1;
        }
        return ans;
    }

```